Universidade Federal do ABC
Fundamentos de Processamento Gráfico 2023.3
Prof. Celso Setsuo Kurashima
Andressa Guimaraes Benedicto 11201810280
Heitor Rodrigues Savegnago 11077415
Kaleb Lucas da Silva Alves 21049916
Pedro Domingos Napole Certo 11201722682

Roteiro - Projeto Final

1- Introdução

Este roteiro pode ser usado como base de apoio para a utilização do software de visualização dos modelos atômicos durante o decaimento. A linguagem utilizada no programa é o OpenGL e foram utilizados na elaboração diversos conceitos aprendidos em sala de aula durante o quadrimestre.
O arquivo do programa está disponível juntamente com esse relatório, é necessário baixar a pasta como zip e descompactá-la nos arquivos locais.

2 - Fundamentos Iniciais

Quando os números de prótons e de nêutrons no núcleo de um isótopo não confere estabilidade, o átomo passa por um processo conhecido como decaimento radioativo. Nesse evento, o núcleo emite radiações alfa, beta e gama e aquele átomo se torna de outro elemento. O software em questão, demonstra o decaimento da cadeia do Urânio 235 até o chumbo. Nele se pode ver uma renderização desse evento de forma simplificada, em que o objetivo principal não é uma representação fidedigna das proporções atômicas, mas sim, a apresentação de uma cadeia de decaimento atômico de maneira didática.

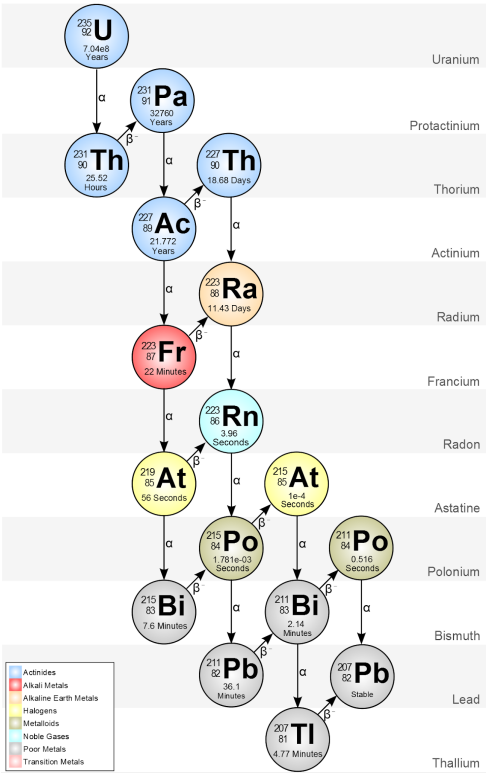


Figura 1 - decaimento do Urânio

3- Rodando o programa

Abrindo o terminal na pasta em que a pasta com o programa está armazenado, use o seguinte comando para o compilar:

```
gcc projeto.cpp -o projeto -lSOIL -lglut -lGl -lGLU -lm
```

Certifique-se de que à biblioteca SOIL está baixada no computador que está utilizando rodando o programa

Levando em consideração a extensão do código, não analisaremos ele por completo, mas comentaremos nesta seção a funcionalidade das funções declaradas no programa. O código na íntegra está anexado à entrega do relatório.

4- Código do programa

Levando em consideração a extensão do código, não analisaremos ele por completo, mas comentaremos nesta seção a funcionalidade das funções declaradas no programa. O código na íntegra está anexado à entrega do relatório.

4.1 Blocos de declaração

Bloco de declaração de tipos especiais

Nele é declarado uma struct que conterá as informações referentes a cada um dos átomos.O primeiro espaço é denominado ‘elemento’ que é por onde haverá a identificação do elemento que estamos lidando. Além disso, temos mais cinco espaços, declarados como int, estes são, respectivamente, um para o número de prótons e um de nêutrons, índice do elemento associado ao decaimento alfa, índice do elemento associado ao decaimento beta e o raio atômico de valência (o raio mais externo dos elétrons associados).
Há também a declaração de um ‘enum’ denominado ‘Electronic Layers’ que usaremos mais à frente para definir a quantidade de elétrons que haverá em cada camada.

Bloco de declaração de constantes

Nesse bloco são declaradas diversas constantes, a começar com uma constante ‘inválida’ (INV = -1) que utilizaremos no decorrer do código. São declaradas em seguida, constantes referentes às cores dos prótons, dos nêutrons e dos elétrons. Também são definidos uma cor branca como referência, o nível de saturação, o raio associado ao núcleo do átomo, o fps (frame per second), as coordenadas para os botões alfa e beta, bem como o raio para a circunferência do botão.

Declaramos também uma lista com todos os átomos, seus dados e os índices referentes aos decaimentos alfa e beta, caso não possua, utilizamos a constante INV. Por fim, há uma string que utilizamos como referência para a construção das frases na tela e as strings de referência para os arquivos com os símbolos alfa e beta dos botões.

Bloco de declaração de variáveis

Aqui são declaradas as variáveis globais auxiliares, como as cores relativas da interface, o ângulo para a rotação, o comprimento e largura da janela, índice do átomo atual, um booleano para saber se o átomo

está rotacionando ou não, um contador de camadas, um int para o raio do núcleo de valência, e também índices de textura do objeto e dos botões.

Bloco de declaração de funções

Informa o compilador da existência das funções e nos permite chamá-las independentemente da ordem em que elas se encontram no código.

4.2 Main e inicialização GLUT

main

Utilizamos um ‘set_locale’ para comunicar ao wide char que deve operar na linguagem do local atual do usuário, para que ele aceite símbolos específicos. Chamamos também o init_glut com os argumentos do programa e nomeando a nossa janela. Finalizamos o main com o glutMainLoop() para que o programa execute em ciclos.

void init_glut(const char *window_name, int *argcp, char **argv);

Para a elaboração deste programa nós utilizamos o OpenGL Utility Toolkit (GLUT), que é uma biblioteca de utilitários para programas em OpenGL. Ela facilita na elaboração de formas geométricas, mas também permite que nós tenhamos um maior controle da janela, bem como auxilia no monitoramento de inputs do teclado e do mouse. Nesta função nós inicializamos esta biblioteca.

Dentro dos comandos nós temos a criação e as definições das dimensões e posicionamento da janela inicializada. Damos também o comando para utilização de coloração RGB, single e depth buffering. Nessa mesma função, são definidas as funções callback, que serão utilizadas como resposta a determinados eventos, veremos cada uma em detalhes mais a frente. Vale um destaque para a função timer, vista em aula, que garante a renderização dos frames na frequência correta; Ela recebe como argumento a quantidade de milissegundos de intervalo a cada chamada da função, qual função ela deve chamar e o argumento que deve ser passado à função.

É criado um menu interativo que será controlado pelo mouse e será ativado pelo clique do botão direito (glutAttachMenu). Definimos a iluminação e após inserimos duas funções, setElectronicLayers, que é responsável por definir quais elétrons devem ficar em cada camada, e loadTexture, que puxar a primeira das imagens que aparecerão no canto da tela como referência para cada um dos elementos.

Por fim, utilizamos SOIL_load_OGL_texture para carregar a textura de alfa e beta para os botões que podem ser utilizados para seguir o próximo decaimento.

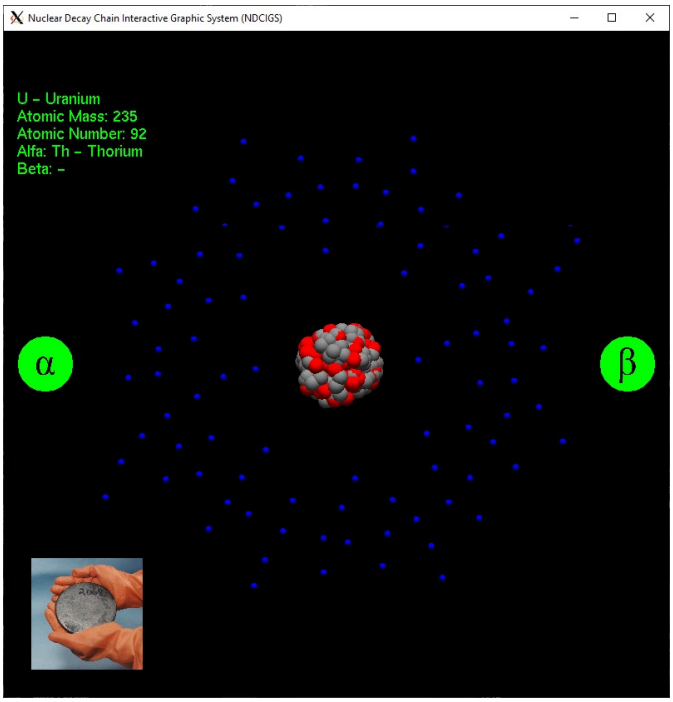
4.3 Funções callback

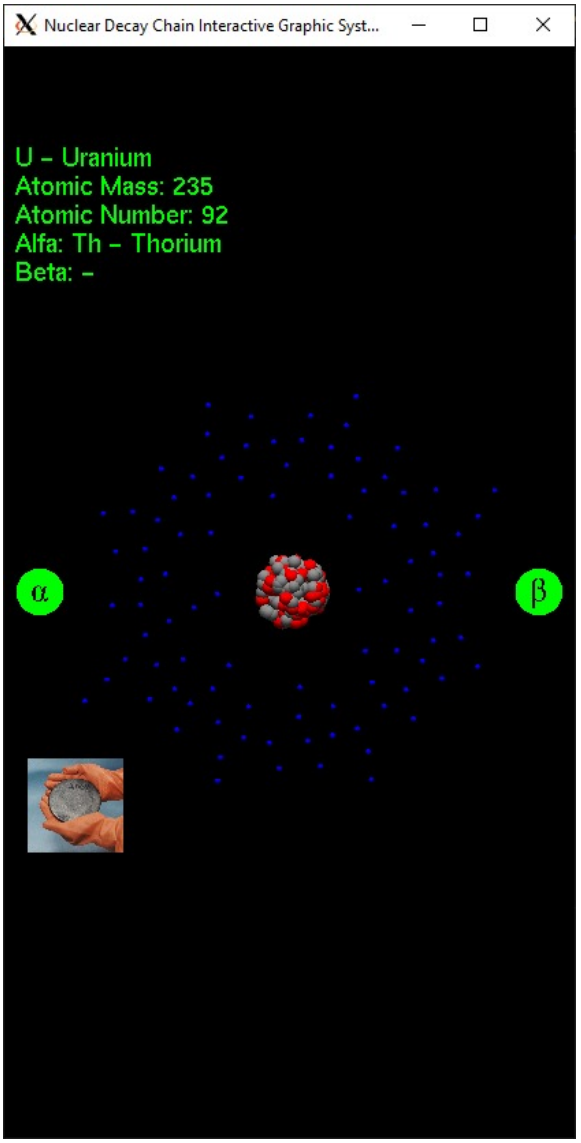
void display_callback(void);

Essa função é chamada para ‘reinicializar’ o display. Nela está contido um glClear para limpar os buffers, são chamadas as funções responsáveis pela textura, por desenhar o objeto e pelo texto e por fim um glutSwapBuffers para que a imagem renderizada apareça.

void reshape_callback(int w, int h);

A reshape_callback é responsável pelo processamento do objeto quando a janela passa por um reshape. Essa função é necessária pois o objeto não pode ter um tamanho estático, é necessário que ele se adapte às proporções presentes do display. Ela vai receber as dimensões atuais da tela como argumento, redefinir à zona da janela onde vai se desenhar, vai mudar para o modo Projection e reiniciar a projeção. Uma das partes principais da função é um bloco if-else que analisa a largura com relação a altura e cria uma matriz ortográfica, ela vai garantir que o desenho do átomo se mantenha no centro da tela. Ela é uma função interessante pois se a largura tiver uma discrepância muito grande com a altura ele atribui como referência à menor das dimensões. Ajusta a perspectiva para estar olhando para o átomo, retorna ao modo ModelView e limpa a identidade anterior.





Acima podemos ver como o desenho do átomo não é afetado com o redimensionamento da janela graças a função reshape_callback().

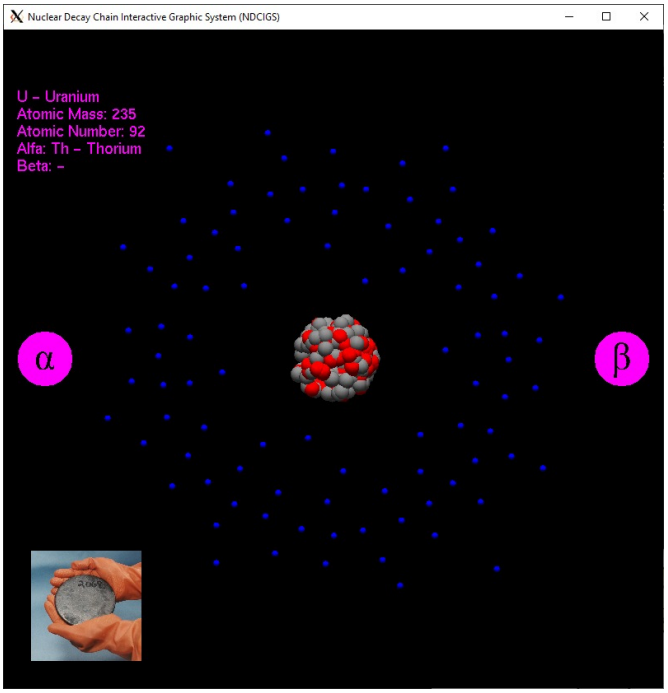
void keyboard_callback(unsigned char key, int x, int y);

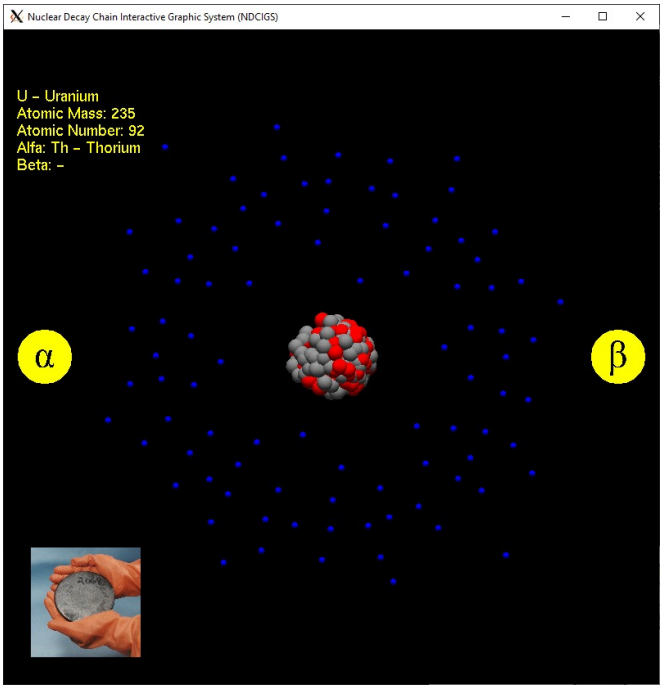
Permite que usuário utilize o teclado para interagir com o programa, utilizando ‘switch’ nós conseguimos desenhar diversas possibilidades de controle através de teclas específicas, como a seleção dos decaimentos alfa (‘a’) ou beta(‘b’), a reinicialização da simulação(‘u’), iniciar a rotação do objeto(‘r’),

parar a rotação do objeto(‘s’), aumentar a velocidade de rotação (‘+’ ou ‘=’), diminuir a velocidade de rotação (‘-’ ou ‘_’) e o fechamento do programa(‘esc’).

void keyboard_callback_special(int key, int x, int y);

Similar à função anterior em estrutura, mas essa permite que o usuário, utilizando as teclas de F1 a F7 altere a cor da interface, sendo cada uma das teclas relativa a uma cor diferente, como exemplificado nas imagens abaixo.





void mouse_callback(int button, int state, int x, int y);

Diferentemente das funções anteriores, esta lida com a interação do usuário através do mouse. Ao clicar com o botão direito a imagem inicia uma animação de rotação. Contudo, ao clicar no botão esquerdo, o usuário encontra um pequeno menu interativo, para o funcionamento apropriado, foi necessário a criação de uma variável que contivesse as coordenadas do mouse dentro da tela.

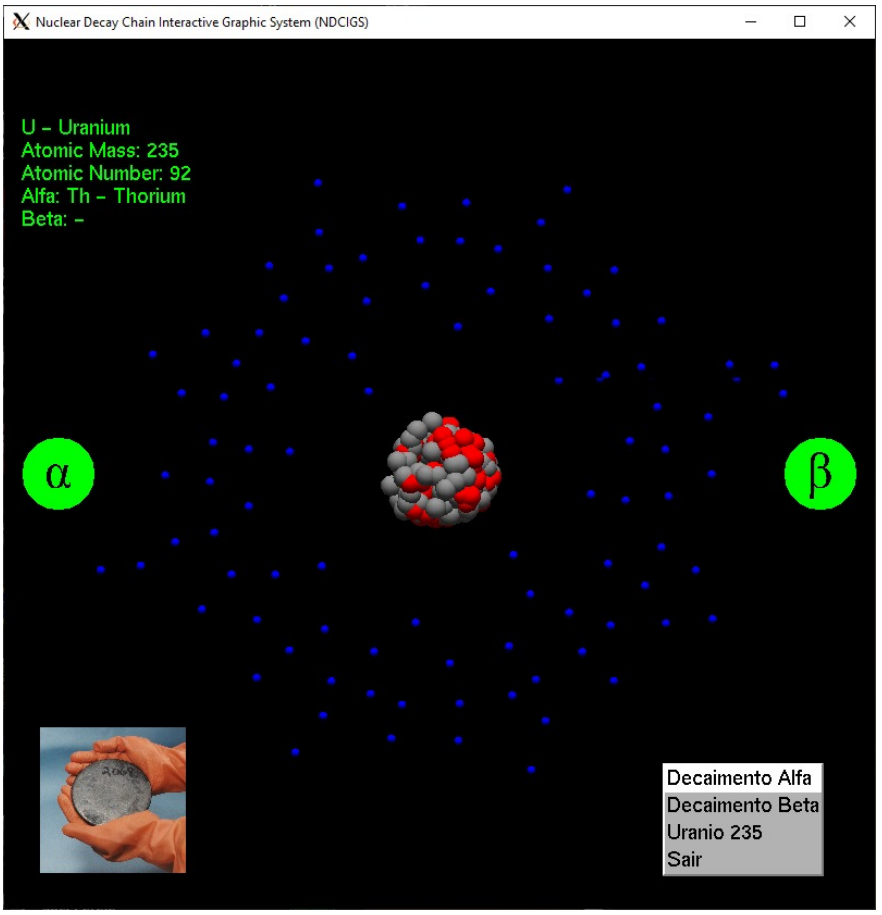
Observa-se que para essa função foi utilizada análise de colisão, comparando a posição do mouse às posições dos botões. Para isso foi necessário à realização de um cálculo relativo Assim, mapeando as coordenadas do mouse, que é relativa ao espaço da tela e varia de acordo com o reshape, o programa identifica qual foi o botão que o usuário apertou (closer_than) e toma as ações referentes àquele botão específico. Temos duas opções, decaimento alfa e decaimento beta, se o usuário clica longe de ambos, como default é retornada a animação da rotação, caso o usuário clique com o botão do meio é pausada à rotação.

void timer_callback(int i);

O timer_callback trabalha em conjunto com a função spinDisplay para fazer a animação dos elétrons em que eles parecem tremer na tela. A cada intervalo de tempo a função spinDisplay é chamada novamente e ocorre uma pequena rotação na posição dos elétrons.

void menu_callback(int value);

Função que contém os comandos referentes ao menu interativo. Usando um switch temos quatro opções de ação, chamada da função para o decaimento alfa, chamada da função para o decaimento beta, reinicialização da simulação e saída do programa.
Termina com uma função necessária para a animação, glutPostRedisplay();, ela marca que o plano normal da janela atual precisa passar por um redisplay, assim conseguimos redesenhar o ecrã em cada frame.



Acima, uma demonstração do layout do menu interativo

4.4 Funções para desenhar os objetos

void draw_object(void);

Essa função é responsável pela renderização da imagem do objeto. Começamos com a definição da quantidade de prótons, nêutrons e o raio atômico como referência, bem como a quantidade de elétrons já impressos na tela para que não haja gastos extras com consulta de memória. Com as informações relativas ao índice do átomo que será desenhado, a função utiliza três loops ‘for’ para desenhar as circunferências que farão referência aos números de prótons, nêutrons e elétrons das camadas do átomo.

Dentro de cada ‘for’ temos um pushMatrix para que entre em um espaço limpo, em seguida é aplicado um Rotatef que determinará a posição dos prótons e dos nêutrons por um ângulo aleatório. O Translatef, que recebe como parâmetro o raio atual do núcleo e o posiciona numa posição aleatória à essa distância. Em seguida é utilizado um popMatrix para que a próxima bolinha seja desenhada no lugar certo.

Para os elétrons, as circunferências que os representam, são posicionadas de modo aleatório dentro de cada camada usando comandos similares aos prótons e os nêutrons, mas levando em consideração a quantidade de elétrons em cada camada, bem como o distanciamento deles do núcleo. O ângulo é determinado levando em consideração a rotação do átomo e há a utilização de funções randômicas na

rotação e na translação para passar à ideia de vibração dos elétrons. Depois que o elétron é desenhado, o popMatrix é chamado e é atualizada a contagem da quantidade de elétrons desenhados.
Fazemos então uma comparação com a quantidade de prótons desenhados, isso se dá para que o número de elétrons desenhados possa ser relativo a quantidade de prótons no núcleo, como uma tentativa de uma representação gráfica mais fidedigna à realidade.

void draw_interface(float radius);

Função responsável pela exibição de todos os elementos gráficos que não fazem parte do átomo. Utilizando a função swprintf conseguimos desenhar na tela as informações que estão na parte superior esquerda. A utilização da biblioteca wchar_t viabiliza a utilização de caracteres especiais, como letras acentuadas. É utilizado um render_text que exibirá o texto na tela.
Em seguida temos um bloco responsável pela exibição dos botões que controlam o decaimento , ele vai desenhar dois botões a partir das informações da coordenada, raio do botão e a textura que será impressa. Por fim, um bloco para a exibição da imagem que aparecerá no canto inferior esquerdo da interface, referente a cada elemento. Ele carrega a figura como textura e atribui à um retângulo desenhado no canto da tela.

void spinDisplay(void);

Ela controla a rotação do átomo, verificando se a rotação está ativada ou não. Caso esteja, ela aumenta o ângulo e se ele for superior a 360° ela o reseta. A função é finalizada por um redisplay através da utilização de glutPostRedisplay.

void draw_particle(float radius);

Utiliza a função glutSolidSphere para desenhar uma esfera utilizando o raio que recebe como parâmetro. É chamada nas funções draw_neutron, draw_eletron e draw_proton.

void draw_eletron(void); / void draw_proton(void); / void draw_neutron(void);

As três funções são similares, sendo diferente o valor do parâmetro passado como o raio para a função draw_particle, que representa o tamanho da partícula desenhada, bem como a iluminação daquela partícula, pois elas são impressas com cores distintas (relative_color). Ela é dividida em duas partes, a primeira contém as funções do OpenGL usadas para determinar a iluminação do objeto. Após a chamada da função draw_particle, ocorre o reset dessas propriedades que foram determinadas para evitar que os próximos objetos renderizados saiam com as mesmas propriedades do material.

void draw_circle(const GLfloat c[2], float r, GLint tex);

Desenha um círculo na tela com base na coordenada, usando também a textura e o raio recebidos como parâmetro. Ela começa desenhando um círculo na coordenada específica com o raio passado, ativa a textura transparente, desenha um quadrado ao redor do círculo e por fim, aplica à textura em cima do quadrado.

4.5 Funções de ações

void alfa_decay(void)/void beta_decay(void);

Responsáveis pelos decaimentos, verificando se o elemento possui um decaimento daquele tipo ou se o índice é inválido, caso ele tenha, ocorre a atualização do índice do átomo, da textura da tela e dos elétrons associados ao elemento.

bool closer_than(const GLfloat c_A[2], const GLfloat c_B[2], const GLfloat d);

Ela é chamada na função 'mouse_callback', compara a distância de um ponto A e um ponto B até um ponto específico que é passada e retorna verdadeiro ou falso, a partir dessa comparação.

void render_text(const char *text, int x, int y);

Ela recebe como parâmetro o texto que será exposto, esse texto é widechar então podemos utilizar caracteres especiais, bem como o posicionamento desse texto pelas coordenadas x e y.
Utilizando um 'for' o texto é lido e renderizado. Para isto, foi utilizada a função BitmapCharacter, da biblioteca GLUT. Tivemos que levar em consideração também a quebra de linha, e como o OpenGL não é uma linguagem voltada para textos, a quebra de linha é feita com o ajuste da posição em que o texto está sendo impresso todas as vezes que for lido '\n'.

void loadTexture(int elementIndex);

Ela verifica o átomo atual, se baseando em seu número de prótons, e invoca o arquivo de imagem associado. Vale ressaltar que o nome dos arquivos seguem o seguinte formato '(número atômico do elemento).png'. Ao carregar ela associa o arquivo à variável texture.

void setElectronicLayers(int Z);

Verifica a quantidade de prótons do átomo e vai fazer a distribuição dos elétrons seguindo a lógica do diagrama de Linus Pauling. Essa função foi adicionada para que os elétrons fiquem distribuídos numa quantidade apropriada por camadas. Eles são, portanto, associados à camadas específicas no bloco if-else, considerando que se há elétrons faltando naquela camada, o elétron será adicionado à ela.

Referências:

- NEIDER, Jackie; DAVIS, Tom; WOO, Mason. **OpenGL programming guide**. Reading, MA: Addison-Wesley, 1993.
- Uranium 235 and Uranium 238. **Washington University in St. Louis.** Disponível em <https://sites.wustl.edu/hazardouswaste/radionuclides/uranium/>. Acesso em 11 de dezembro de 2023.